

1 What is claimed is:

1 1. A method for dynamic transformation of programs, said method operable at
2 least in part within an information processing system, comprising:

3 a. accessing a dynamic instrumentation interface of a mixed mode runtime
4 environment; and

5 b. controlling a compiler of the mixed mode runtime environment via the
6 interface to transform a program.

1 2. The method of claim 1, wherein the mixed mode runtime environment is a
2 virtual machine and the compiler is a just in time (JIT) compiler, step a
3 further comprising determining whether a first class is loaded, and step b
4 comprising, if the first class is loaded, controlling the JIT compiler to compile
5 the first class into an executable program and transform the executable
6 program into a transformed program.

1 3. The method of claim 2, wherein the virtual machine is one of the group of a
2 Java virtual machine and a .NET virtual machine, further comprising, if the
3 first class is not loaded, storing transformation information relating to the first
4 class and monitoring the virtual machine to determine when the first class is
5 loaded.

1 4. The method of claim 3, wherein the steps of monitoring and determining
2 when the first class is loaded are performed by a class loader responsive to
3 instructions via the interface.

1 5. The method of claim 2, wherein the virtual machine is one of the group of a
2 Java virtual machine and a .NET virtual machine, further comprising, if the
3 first class is loaded, allowing a current instantiation of the first class to run
4 until terminated.

1 6. The method of claim 2, wherein the virtual machine is one of the group of a
2 Java virtual machine and a .NET virtual machine, further comprising, when

3 the first class is already loaded, replacing currently running code based on an
4 old-object method of the first class with the transformed program and
5 adjusting an activation stack so existing invocations of the old-object method
6 continue executing after adjusting the activation stack.

1 7. The method of claim 1, wherein step a comprises initiating an
2 instrumentation client program, the step of accessing comprises making a
3 predetermined transformation call to the interface, and step b. comprises the
4 instrumentation client controlling the compiler via the call to the interface.

1 8. The method of claim 7, wherein the predetermined call is one of a group
2 consisting of: *InsertNewBytecodesBefore*; *InsertNewBytecodesAfter*;
3 *DeleteBytecodes*; *ReplaceClassfile*; *ReplaceMethod*; *InsertJNlCallBefore*; and.
4 *InsertJNlCallAfterclass*.

1 9. An information handling system comprising a processor, a mixed mode
2 virtual machine (VM) and a dynamic instrumentation interface operably
3 coupling the VM and a program instrumentation tool, the VM comprising a
4 compiler operably coupled to the interface and responsive to signaling from the
5 interface to transform a program.

1 10. The system of claim 9, wherein the compiler comprises a just in time (JIT)
2 compiler, the VM comprises plural instructions and the processor is operably
3 configured to execute said plural instructions, the plural instructions
4 comprising:
5 class loader instructions operable for determining whether a first class is
6 loaded, and when the first class is loaded, providing first class information
7 from a class loader to the JIT compiler; and
8 transformation instructions operable for controlling the JIT compiler to
9 compile the first class into a processor executable program and to transform
10 the executable program into a transformed program.

1 11. The system of claim 10, wherein the VM is one of a group of a Java virtual
2 machine and a .NET virtual machine, and the class loader instructions are
3 further operable, when the first class is not loaded, to store transformation
4 information relating to the first class and monitor the VM to determine when
5 the first class is loaded.

1 12. The system of claim 11, wherein the class loader instructions for
2 monitoring and determining when the first class is loaded are performed by a
3 class loader responsive to first transformation signaling via the interface.

1 13. The system of claim 10, wherein the VM is one of the group of a Java
2 virtual machine and a .NET virtual machine, the transformation instructions
3 comprising further instructions operable for, when the first class is loaded,
4 allowing a current instantiation of the first class to run until terminated.

1 14. The system of claim 13, further comprising operating instructions
2 independent of the VM, wherein the transformed program is operably executed
3 by the operating instructions independent of the VM.

1 15. The system of claim 10, the transformation instructions comprising
2 further instructions configured to, when the first class is already loaded,
3 replace currently running code based on an old-object method of the first class
4 with the transformed program and adjust an activation stack so existing
5 invocations of the old-object method continue executing after adjusting the
6 activation stack.

1 16. The system of claim 9, further comprising a client tool program having
2 client transformation instructions operable for initiating a transformation
3 request to the interface via a predetermined transformation call, thereby
4 controlling the compiler via the call to the interface.

1 17. The system of claim 16, wherein the interface comprises further
2 instructions responsive to the predetermined call, the call being one of a group
3 consisting of: *InsertNewBytecodesBefore*; *InsertNewBytecodesAfter*;

4 *DeleteBytecodes; ReplaceClassfile; ReplaceMethod; InsertJNlCallBefore; and.*
5 *InsertJNlCallAfterclass.*

1 18. A program product in a signal bearing medium executable by a device for
2 presenting a hierarchical representation of a target program, the product
3 comprising:

4 VM instructions operable as a mixed-mode virtual machine (VM) comprising a
5 compiler;

6 interface instructions operable as a dynamic instrumentation interface for
7 coupling the VM and a program instrumentation tool, further operable
8 responsive to signaling from the interface to transform a program being
9 operated on by the VM.

1 19. The program product of claim 18, wherein the compiler is operable as a
2 just in time (JIT) compiler, the VM instructions further comprising:

3 class loader instructions operable for determining whether a first class is
4 loaded, and when the first class is loaded, providing first class information
5 from a class loader to the JIT compiler; and

6 transformation instructions operable for controlling the JIT compiler to
7 compile the first class into a processor executable program and to transform
8 the executable program into a transformed program.

1 20. The program product of claim 19, wherein the VM is operable as one of a
2 group of a Java virtual machine and a .NET virtual machine, and the class
3 loader instructions are further configured to, when the first class is not loaded,
4 store transformation information relating to the first class and monitor the
5 VM to determine when the first class is loaded.

1 21. The program product of claim 20, wherein the class loader instructions for
2 monitoring and determining when the first class is loaded are configured to be

3 performed by a class loader responsive to first transformation signaling via the
4 interface.

1 22. The program product of claim 19, wherein the VM is operable as one of the
2 group of a Java virtual machine and a .NET virtual machine, the
3 transformation instructions comprising further instructions configured to,
4 when the first class is loaded, permit a current instantiation of the first class
5 to run until terminated.

1 23. The program product of claim 22, wherein the transformed program is
2 configured to be operably executed by operating instructions independent of
3 the VM.

1 24. The program of claim 19, the transformation instructions comprising
2 further instructions configured to, when the first class is already loaded,
3 replace currently running code based on an old-object method of the first class
4 with the transformed program and adjust an activation stack so existing
5 invocations of the old-object method continue executing after adjusting the
6 activation stack.

1 25. The program product of claim 18, further comprising client transformation
2 instructions operably part of a client tool and configured to initiate a
3 transformation request to the interface via a predetermined transformation
4 call, thereby controlling the compiler via the call to the interface.

1 26. The program product of claim 25, wherein the interface instructions are
2 further configured to be responsive to the predetermined transformation call,
3 the call being one of a group consisting of: *InsertNewBytecodesBefore*;
4 *InsertNewBytecodesAfter*; *DeleteBytecodes*; *ReplaceClassfile*; *ReplaceMethod*;
5 *InsertJNlCallBefore*; and. *InsertJNlCallAfterclass*.